

autogluon_each_location

November 3, 2023

1 Config

```
[45]: # config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*10
presets = "best_quality" #'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = [] #["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

drop_night_outliers = True
drop_null_outliers = False

# to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",
# ↪ "dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",
# ↪ "wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",
# ↪ "rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
# ↪ gm3", "air_density_2m:kgm3"]#, "msl_pressure:hPa", "pressure_50m:hPa",
# ↪ "pressure_100m:hPa"]
to_drop = ["wind_speed_w_1000hPa:ms", "wind_speed_u_10m:ms", "wind_speed_v_10m:
↪ ms"]

excluded_model_types = ['CAT', 'XGB', 'RF']

use_groups = False
n_groups = 8

# auto_stack = True
num_stack_levels = 0
num_bag_folds = None# 8
num_bag_sets = None#20
```

```

use_tune_data = True
use_test_data = True
#tune_and_test_length = 0.5 # 3 months from end
# holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val_
↳and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False#
sample_weight_estimated = 1
sample_weight_may_july = 1

run_analysis = False

shift_predictions_by_average_of_negatives_then_clip = False
clip_predictions = True
shift_predictions = False

```

2 Loading and preprocessing

```

[46]: import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings("ignore")

def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,
    ↳we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")

    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
               'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
               'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    # Filter rows where index.minute == 0
    X_shifted = X[X.index.minute == 0][columns].copy()

    # Create a set for constant-time lookup
    index_set = set(X.index)

```

```

# Vectorized time shifting
one_hour = pd.Timedelta('1 hour')
shifted_indices = X_shifted.index + one_hour
X_shifted.loc[shifted_indices.isin(index_set)] = X.
↳loc[shifted_indices[shifted_indices.isin(index_set)]] [columns]

# set last row to same as second last row
X_shifted.iloc[-1] = X_shifted.iloc[-2]

# Count
count1 = len(shifted_indices[shifted_indices.isin(index_set)])
count2 = len(X_shifted) - count1

print("COUNT1", count1)
print("COUNT2", count2)

# Rename columns
X_old_unshifted = X_shifted.copy()
X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
↳columns]

date_calc = None
# If 'date_calc' is present, handle it
if 'date_calc' in X.columns:
    date_calc = X[X.index.minute == 0]['date_calc']

# resample to hourly
print("index: ", X.index[0])
X = X.resample('H').mean()
print("index AFTER: ", X.index[0])

X[columns] = X_shifted[columns]
#X[X_old_unshifted.columns] = X_old_unshifted

if date_calc is not None:
    X['date_calc'] = date_calc

return X

def fix_X(X, name):

```

```

    # Convert 'date_forecast' to datetime format and replace original column
    ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)

    X = feature_engineering(X)

    return X

def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")

    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated

    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train

def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,
    ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =
    ↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -
    ↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600

```

```

X_test["estimated_diff_hours"] = (X_test.index - pd.
↳to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

X_train_estimated["estimated_diff_hours"] =
↳X_train_estimated["estimated_diff_hours"].astype('int64')
    # the filled once will get dropped later anyways, when we drop y nans
X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
↳fillna(-50).astype('int64')

if use_is_estimated_attr:
    X_train_observed["is_estimated"] = 0
    X_train_estimated["is_estimated"] = 1
    X_test["is_estimated"] = 1

# drop date_calc
X_train_estimated.drop(columns=['date_calc'], inplace=True)
X_test.drop(columns=['date_calc'], inplace=True)

y_train["y"] = y_train["pv_measurement"].astype('float64')
y_train.drop(columns=['pv_measurement'], inplace=True)
X_train = pd.concat([X_train_observed, X_train_estimated])

# clip all y values to 0 if negative
y_train["y"] = y_train["y"].clip(lower=0)

X_train = pd.merge(X_train, y_train, how="inner", left_index=True,
↳right_index=True)

# print number of nans in y
print(f"Number of nans in y: {X_train['y'].isna().sum()}")

print(f"Size of estimated after dropping nans:
↳{len(X_train[X_train['is_estimated']==1].dropna(subset=['y']))}")

X_train["location"] = location
X_test["location"] = location

return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []

```

```

# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed = pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,
    ↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)

```

```

Processing location A...
COUNT1 29667
COUNT2 1
index: 2019-06-02 22:00:00
index AFTER: 2019-06-02 22:00:00
COUNT1 4392
COUNT2 2
index: 2022-10-28 22:00:00
index AFTER: 2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index: 2023-05-01 00:00:00
index AFTER: 2023-05-01 00:00:00
Number of nans in y: 0
Size of estimated after dropping nans: 4418
Processing location B...
COUNT1 29232
COUNT2 1
index: 2019-01-01 00:00:00
index AFTER: 2019-01-01 00:00:00
COUNT1 4392
COUNT2 2

```

```

index: 2022-10-28 22:00:00
index AFTER: 2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index: 2023-05-01 00:00:00
index AFTER: 2023-05-01 00:00:00
Number of nans in y: 4
Size of estimated after dropping nans: 3625
Processing location C...
COUNT1 29206
COUNT2 1
index: 2019-01-01 00:00:00
index AFTER: 2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index: 2022-10-28 22:00:00
index AFTER: 2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index: 2023-05-01 00:00:00
index AFTER: 2023-05-01 00:00:00
Number of nans in y: 6059
Size of estimated after dropping nans: 2954

```

2.1 Feature engineering

2.1.1 Remove anomalies

```

[47]: import numpy as np
import pandas as pd

# loop thorough x train[y], keep track of streaks of same values and replace
↳ them with nan if they are too long
# also replace nan with 0

import numpy as np

def replace_streaks_with_nan(df, max_streak_length, column="y"):
    for location in df["location"].unique():
        x = df[df["location"] == location][column].copy()

        last_val = None
        streak_length = 1
        streak_indices = []
        allowed = [0]
        found_streaks = {}

```

```

for idx in x.index:
    value = x[idx]
    # if location == "B":
    #     continue

    if value == last_val and value not in allowed:
        streak_length += 1
        streak_indices.append(idx)
    else:
        streak_length = 1
        last_val = value
        streak_indices.clear()

    if streak_length > max_streak_length:
        found_streaks[value] = streak_length

        for streak_idx in streak_indices:
            x[idx] = np.nan
            streak_indices.clear() # clear after setting to NaN to avoid
↪setting multiple times
        df.loc[df["location"] == location, column] = x

    print(f"Found streaks for location {location}: {found_streaks}")

return df

# deep copy of X_train into x_copy
X_train = replace_streaks_with_nan(X_train.copy(), 3, "y")

```

Found streaks for location A: {}

Found streaks for location B: {3.45: 28, 6.9: 7, 12.9375: 5, 13.8: 8, 276.0: 78, 18.975: 58, 0.8625: 4, 118.1625: 33, 34.5: 11, 183.7125: 1058, 87.1125: 7, 79.35: 34, 7.7625: 12, 27.6: 448, 273.41249999999997: 72, 264.78749999999997: 55, 169.05: 33, 375.1875: 56, 314.8125: 66, 76.7625: 10, 135.4125: 216, 81.9375: 202, 2.5875: 12, 81.075: 210}

Found streaks for location C: {9.8: 4, 29.400000000000002: 4, 19.6: 4}

```

[48]: # print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],
↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))

```

Dropped rows: 9293

```

[49]: import matplotlib.pyplot as plt
import seaborn as sns

```



```

# Filter out rows where y == 0
temp = X_train[X_train["y"] != 0]

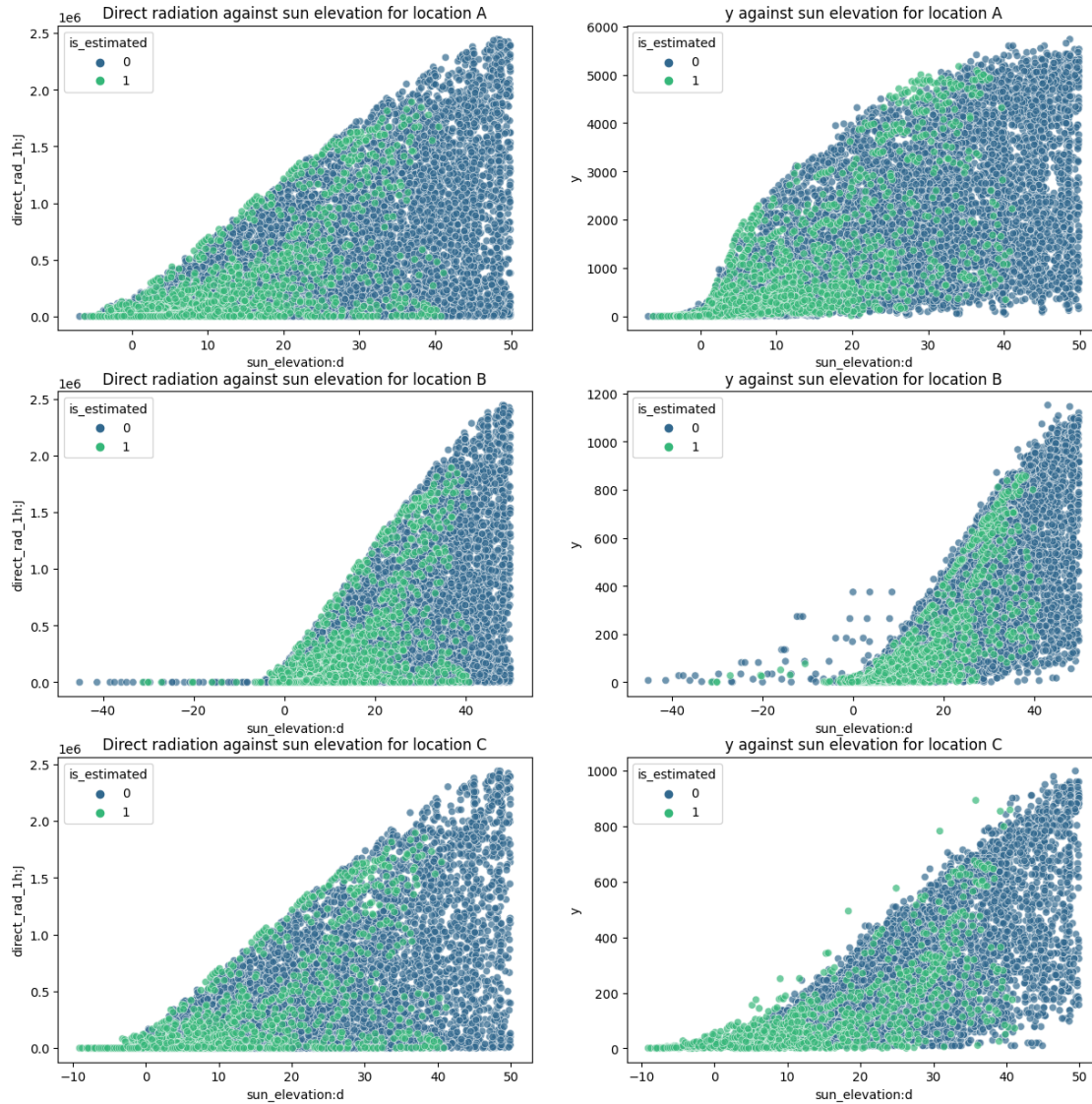
# Plotting
fig, axes = plt.subplots(len(locations), 2, figsize=(15, 5 * len(locations)))

for idx, location in enumerate(locations):
    sns.scatterplot(ax=axes[idx][0], data=temp[temp["location"] == location],
        x="sun_elevation:d", y="direct_rad_1h:J", hue="is_estimated",
        palette="viridis", alpha=0.7)
    axes[idx][0].set_title(f"Direct radiation against sun elevation for
        location {location}")

    sns.scatterplot(ax=axes[idx][1], data=temp[temp["location"] == location],
        x="sun_elevation:d", y="y", hue="is_estimated", palette="viridis", alpha=0.7)
    axes[idx][1].set_title(f"y against sun elevation for location {location}")

# plt.tight_layout()
# plt.show()

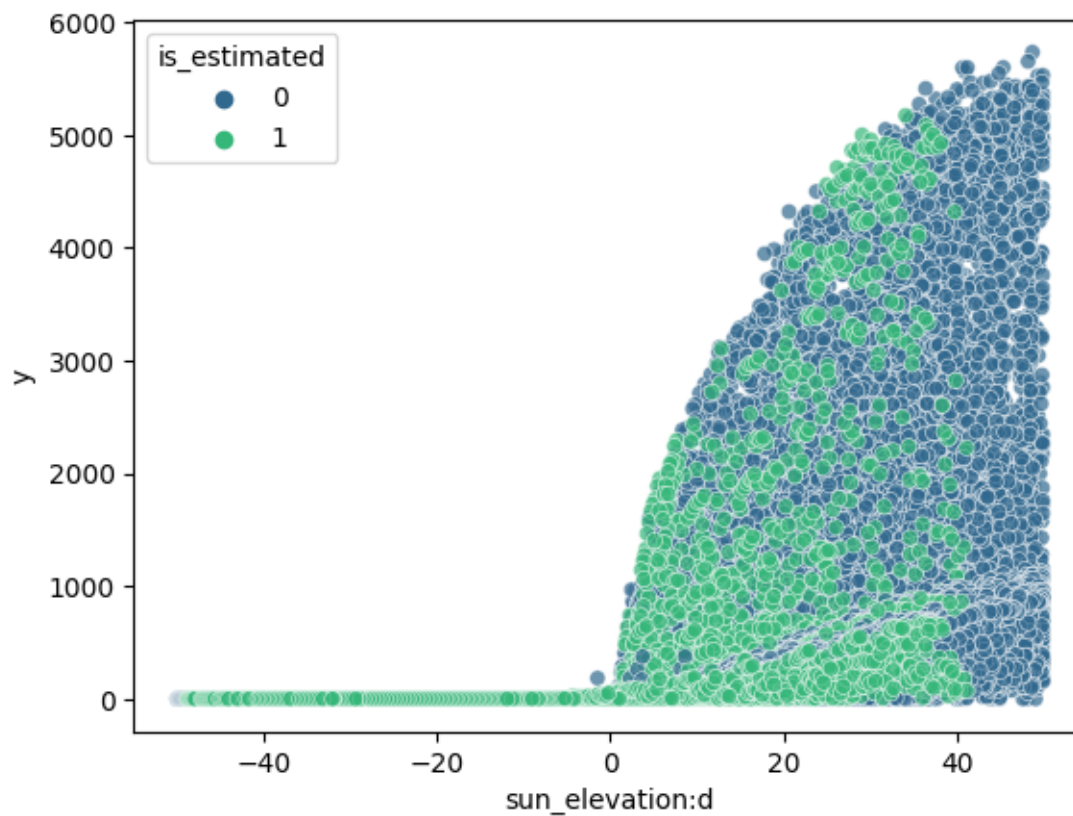
```



```
[50]: thresh = 0.1

# Update "y" values to NaN if they don't meet the criteria
mask = (X_train["direct_rad_1h:J"] <= thresh) & (X_train["diffuse_rad_1h:J"] <=
    ↪ thresh) & (X_train["y"] >= 0.1)
if drop_night_outliers:
    X_train.loc[mask, "y"] = np.nan

# Plot using sns scatterplot
sns.scatterplot(data=X_train, x="sun_elevation:d", y="y", hue="is_estimated",
    ↪ palette="viridis", alpha=0.7)
plt.show()
```

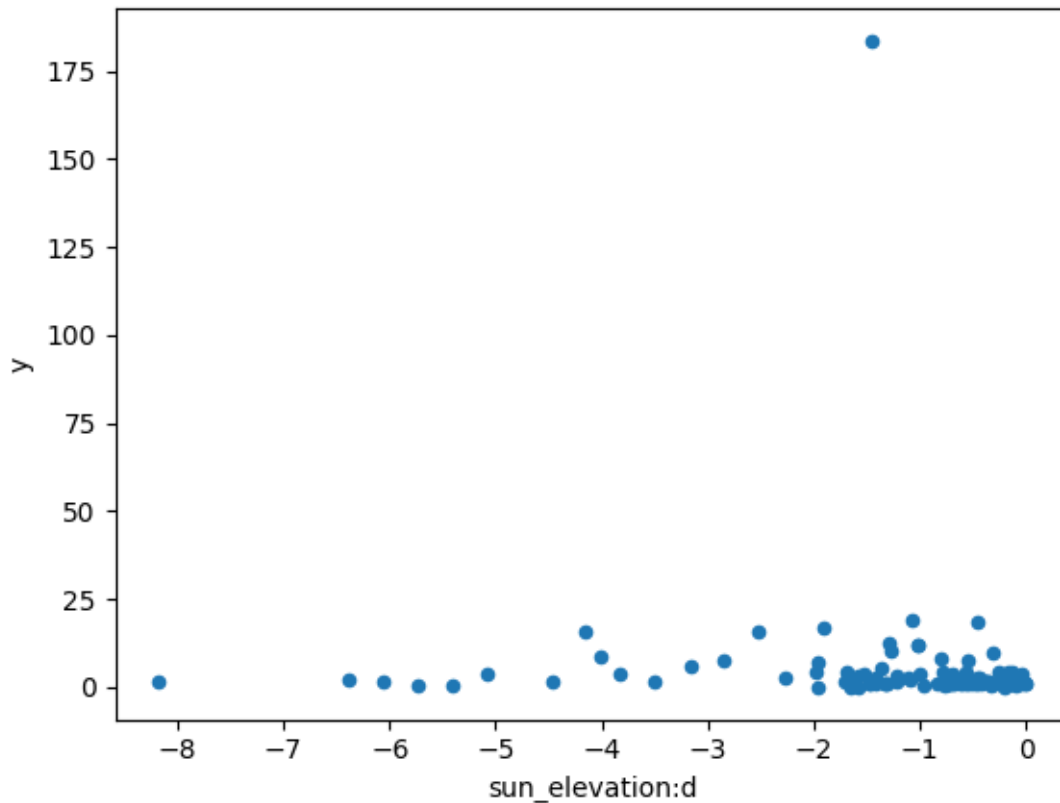


```
[51]: # location B count number of rows with y > 0 and sun_elevation:d < 0
```

```
condition = (X_train["location"] == "B") & (X_train["y"] > 0) & \
    ↪(X_train["sun_elevation:d"] < 0)
bad = X_train[condition]

bad.plot.scatter(x="sun_elevation:d", y="y")
```

```
[51]: <AxesSubplot: xlabel='sun_elevation:d', ylabel='y'>
```



```
[52]: # set y to nan where y is 0, but direct_rad_1h:J or diffuse_rad_1h:J are > 0
      ↪ (or some threshold)
threshold_direct = X_train["direct_rad_1h:J"].max() * 0.01
threshold_diffuse = X_train["diffuse_rad_1h:J"].max() * 0.01
print(f"Threshold direct: {threshold_direct}")
print(f"Threshold diffuse: {threshold_diffuse}")

mask = (X_train["y"] == 0) & ((X_train["direct_rad_1h:J"] > threshold_direct) |
      ↪ (X_train["diffuse_rad_1h:J"] > threshold_diffuse)) & (X_train["sun_elevation:
      ↪ d"] > 0) & (X_train["fresh_snow_24h:cm"] < 6) & (X_train[['fresh_snow_12h:
      ↪ cm', 'fresh_snow_1h:cm', 'fresh_snow_3h:cm', 'fresh_snow_6h:cm']]).
      ↪ sum(axis=1) == 0)
print(len(X_train[mask]))

#print(X_train[mask][[x for x in X_train.columns if "snow" in x]])

# show plot where mask is true
#sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="y",
      ↪ hue="is_estimated", palette="viridis", alpha=0.7)
```

```

sns.scatterplot(data=X_train[mask], x="sun_azimuth:d", y="is_day:idx",
    ↪hue="is_estimated", palette="viridis", alpha=0.7)
plt.show()

#sns.scatterplot(data=X_train[mask], x="fresh_snow_24h:cm",
    ↪y="total_cloud_cover:p", hue="is_estimated", palette="viridis", alpha=0.7)

# plot X_train["y"], but with another color where mask is true (only location B)
fig, ax = plt.subplots()
X_train[X_train["location"] == "B"].plot(y="y", ax=ax, style='.', color='b')
# now scatter plot of mask and b
X_train[(X_train["location"] == "B") & mask].plot(y="y", ax=ax, style='.',
    ↪color='r')

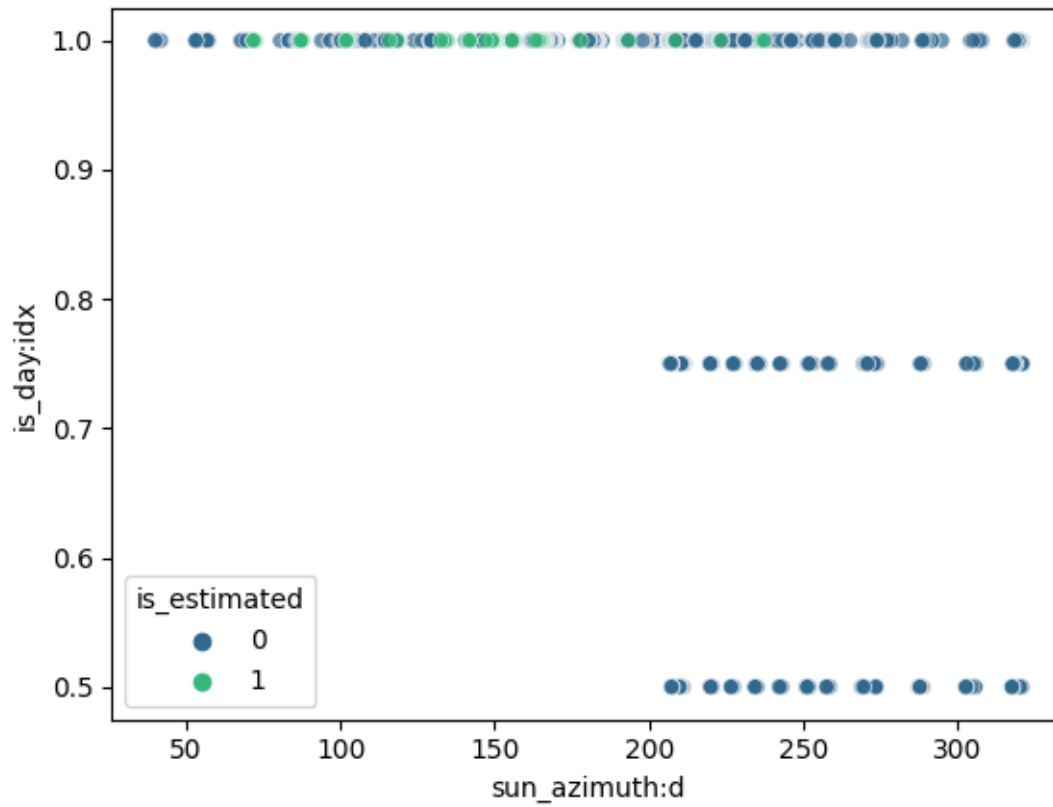
#X_train[(X_train["location"] == "B") & mask].plot(y="y", color='r')

# set y to nan where mask
if drop_null_outliers:
    X_train.loc[mask, "y"] = np.nan

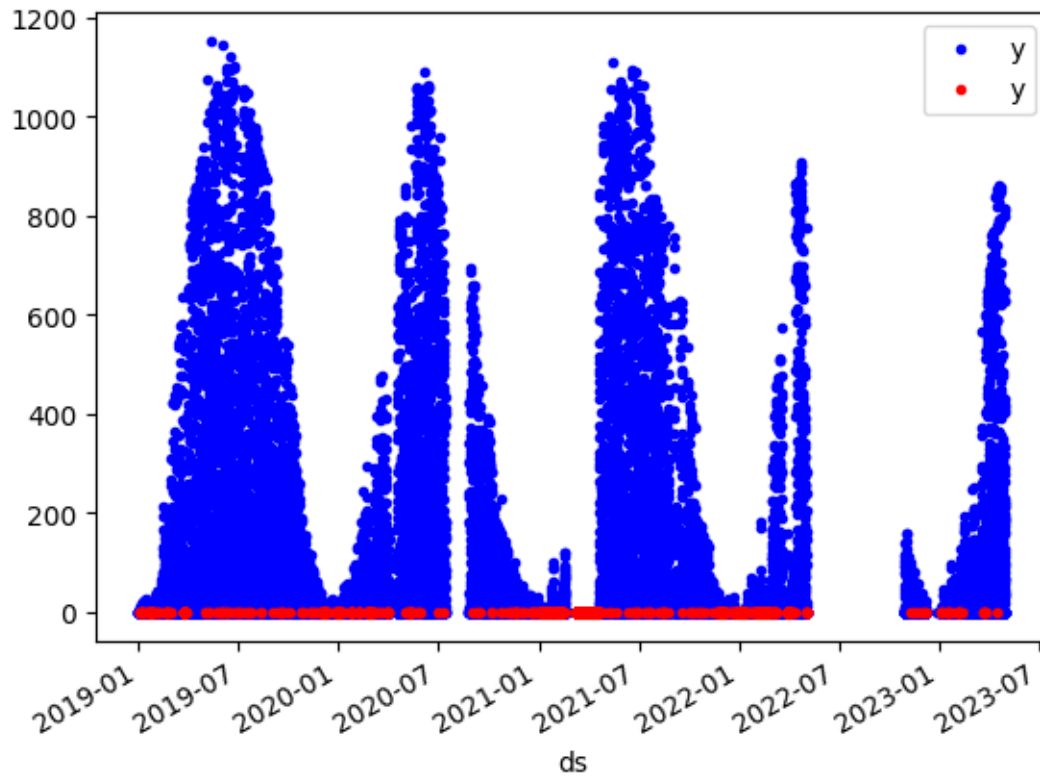
# show how many rows for each location, and for estimated and not estimated
X_train[mask].groupby(["location", "is_estimated"]).count()["direct_rad_1h:J"]

```

Threshold direct: 24458.97
 Threshold diffuse: 11822.5050000000001
 2599



```
[52]: location  is_estimated
      A         0             87
          1             10
      B         0          1250
          1             32
      C         0          1174
          1             46
      Name: direct_rad_1h:J, dtype: int64
```



```
[53]: # print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],
               inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

Dropped rows: 1876

2.1.2 Other stuff

```
[54]: import numpy as np
import pandas as pd

for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

#print(X_train.head())
```

```

# If the "sample_weight" column is present and weight_evaluation is True,
↳ multiply sample_weight with sample_weight_may_july if the ds is between
↳ 05-01 00:00:00 and 07-03 23:00:00, else add sample_weight as a column to
↳ X_train

if weight_evaluation:
    if "sample_weight" not in X_train.columns:
        X_train["sample_weight"] = 1

        X_train.loc[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |
↳ ((X_train.index.month == 7) & (X_train.index.day <= 3)), "sample_weight"] *=
↳ sample_weight_may_july

print(X_train.iloc[200])
print(X_train[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |
↳ ((X_train.index.month == 7) & (X_train.index.day <= 3))].head(1))

if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique() # Assuming 'location' is the name
↳ of the column representing locations

    grouped_dfs = [] # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),
↳ repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())

```



```

X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)

```

absolute_humidity_2m:gm3	7.625
air_density_2m:kgm3	1.2215
ceiling_height_agl:m	3644.050049
clear_sky_energy_1h:J	2896336.75
clear_sky_rad:W	753.849976
cloud_base_agl:m	3644.050049
dew_or_rime:idx	0.0
dew_point_2m:K	280.475006
diffuse_rad:W	127.475006
diffuse_rad_1h:J	526032.625
direct_rad:W	488.0
direct_rad_1h:J	1718048.625
effective_cloud_cover:p	18.200001
elevation:m	6.0
fresh_snow_12h:cm	0.0
fresh_snow_1h:cm	0.0
fresh_snow_24h:cm	0.0
fresh_snow_3h:cm	0.0
fresh_snow_6h:cm	0.0
is_day:idx	1.0
is_in_shadow:idx	0.0
msl_pressure:hPa	1026.775024
precip_5min:mm	0.0
precip_type_5min:idx	0.0
pressure_100m:hPa	1013.599976
pressure_50m:hPa	1019.599976
prob_rime:p	0.0
rain_water:kgm2	0.0
relative_humidity_1000hPa:p	53.825001
sfc_pressure:hPa	1025.699951
snow_density:kgm3	NaN
snow_depth:cm	0.0
snow_drift:idx	0.0
snow_melt_10min:mm	0.0
snow_water:kgm2	0.0
sun_azimuth:d	222.089005
sun_elevation:d	44.503498

```

super_cooled_liquid_water:kgm2          0.0
t_1000hPa:K                             286.700012
total_cloud_cover:p                      18.200001
visibility:m                             52329.25
wind_speed_10m:ms                        2.6
wind_speed_u_10m:ms                      -1.9
wind_speed_v_10m:ms                      -1.75
wind_speed_w_1000hPa:ms                  0.0
is_estimated                             0
y                                         4367.44
location                                 A
Name: 2019-06-11 13:00:00, dtype: object
      absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 23:00:00          7.7          1.2235

      ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
2019-06-02 23:00:00    1689.824951          0.0

      clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 23:00:00          0.0    1689.824951          0.0

      dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  ...  \
ds
2019-06-02 23:00:00    280.299988          0.0          0.0  ...

      t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 23:00:00    286.899994          100.0  33770.648438

      wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 23:00:00          3.35          -3.35

      wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 23:00:00          0.275          0.0

      is_estimated    y  location
ds
2019-06-02 23:00:00          0  0.0          A

[1 rows x 48 columns]

```

```
[55]: # Create a plot of X_train showing its "y" and color it based on the value of
      ↪ the sample_weight column.
      if "sample_weight" in X_train.columns:
          import matplotlib.pyplot as plt
          import seaborn as sns
          sns.scatterplot(data=X_train, x=X_train.index, y="y", hue="sample_weight",
          ↪ palette="deep", size=3)
          plt.show()
```

```
[60]: def normalize_sample_weights_per_location(df):
      for loc in locations:
          loc_df = df[df["location"] == loc]
          loc_df["sample_weight"] = loc_df["sample_weight"] /
          ↪ loc_df["sample_weight"].sum() * loc_df.shape[0]
          df[df["location"] == loc] = loc_df
      return df

import pandas as pd

def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the
    ↪ bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
        frac1 (float): The fraction of each bin to go into the first output
        ↪ dataset.

    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Shuffle the data and split into 2 based on frac1
    np.random.seed(0)
    shuffled_data = input_data.sample(frac=1)
```

```

output_data1 = shuffled_data.iloc[:int(len(input_data) * frac1)]
output_data2 = shuffled_data.iloc[int(len(input_data) * frac1):]

return output_data1, output_data2

# # Calculate bin size
# bin_size = len(input_data) // num_bins

# # Initialize empty DataFrames for output
# output_data1 = pd.DataFrame()
# output_data2 = pd.DataFrame()

# for i in range(num_bins):
#     # Shuffle the data in the current bin
#     np.random.seed(i)
#     current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].
↪ sample(frac=1)

#     # Calculate the sizes for each output set
#     size1 = int(len(current_bin) * frac1)

#     # Split and append to output DataFrames
#     output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
#     output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

# # Shuffle and split the remaining data
# remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)

# remaining_size1 = int(len(remaining_data) * frac1)

# output_data1 = pd.concat([output_data1, remaining_data.iloc[:
↪ remaining_size1]])
# output_data2 = pd.concat([output_data2, remaining_data.
↪ iloc[remaining_size1:]])

# return output_data1, output_data2

```

```

[61]: from autogluon.tabular import TabularDataset, TabularPredictor
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the
↪ first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:

```

```

#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())

# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
    ↳Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
estimated_set = TabularDataset(data[data["ds"] >= split_time]) # only estimated

test_set = pd.DataFrame()
tune_set = pd.DataFrame()
new_train_set = pd.DataFrame()

if not use_tune_data:
    raise Exception("Not implemented")

for location in locations:
    loc_data = data[data["location"] == location]
    num_train_rows = len(loc_data)

    tune_rows = 1500.0 # 2500.0
    if use_test_data:
        tune_rows = 1880.0#max(3000.0,
    ↳len(estimated_set[estimated_set["location"] == location]))

    holdout_frac = max(0.01, min(0.1, tune_rows / num_train_rows)) *
    ↳num_train_rows / len(estimated_set[estimated_set["location"] == location])

    print(f"Size of estimated for location {location}:
    ↳{len(estimated_set[estimated_set['location'] == location])}. Holdout frac
    ↳should be % of estimated: {holdout_frac}")

    # shuffle and split data
    loc_tune_set, loc_new_train_set =
    ↳split_and_shuffle_data(estimated_set[estimated_set['location'] == location],
    ↳40, holdout_frac)
    print(f"Length of location tune set : {len(loc_tune_set)}")
    new_train_set = pd.concat([new_train_set, loc_new_train_set])

    if use_test_data:
        loc_test_set, loc_tune_set = split_and_shuffle_data(loc_tune_set, 40, 0.
    ↳2)
        test_set = pd.concat([test_set, loc_test_set])

```

```

tune_set = pd.concat([tune_set, loc_tune_set])

print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))

if use_groups:
    test_set = test_set.drop(columns=['group'])

tuning_data = tune_set

# number of rows in tuning data for each location
print("Shapes of tuning data", tuning_data.groupby('location').size())

if use_test_data:
    test_data = test_set
    print("Shape of test", test_data.shape[0])

train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of
↳ rows in the training (or tuning) data.
if weight_evaluation:
    # ensure sample weights for data sum to the number of rows in the tuning /
    ↳ train data.
    tuning_data = normalize_sample_weights_per_location(tuning_data)
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)

train_data = TabularDataset(train_data)
tuning_data = TabularDataset(tuning_data)

if use_test_data:

```

```
test_data = TabularDataset(test_data)
```

```
Size of estimated for location A: 4214. Holdout frac should be % of estimated:
0.4461319411485524
Length of location tune set : 1880
Size of estimated for location B: 3533. Holdout frac should be % of estimated:
0.5321256722332296
Length of location tune set : 1880
Size of estimated for location C: 2923. Holdout frac should be % of estimated:
0.6431748203900103
Length of location tune set : 1880
Length of train set before adding test set 77247
Length of train set after adding test set 82277
Shapes of tuning data location
A    1504
B    1504
C    1504
dtype: int64
Shape of test 1128
```

3 Quick EDA

```
[62]: if run_analysis:
        import autogluon.eda.auto as auto
        auto.dataset_overview(train_data=train_data, test_data=test_data,
                                label="y", sample=None)
```

```
[63]: if run_analysis:
        auto.target_analysis(train_data=train_data, label="y", sample=None)
```

4 Modeling

```
[64]: import os

# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for
                                filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
```

```

new_filename += f'_{hello}'

print("New filename:", new_filename)

```

Last submission number: 122
 Now creating submission number: 123
 New filename: submission_123_jorge

```
[65]: predictors = [None, None, None]
```

```

[66]: def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both
    ↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",
        ↪train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]
        ↪== loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",
            ↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
            print("Tune data number of rows:",
            ↪tuning_data[tuning_data["location"] == loc].shape[0])
            if use_test_data:
                print("Test data sample weight sum:",
                ↪test_data[test_data["location"] == loc]["sample_weight"].sum())
                print("Test data number of rows:", test_data[test_data["location"]
                ↪== loc].shape[0])
        predictor = TabularPredictor(
            label=label,
            eval_metric=metric,
            path=f"AutogluonModels/{new_filename}_{loc}",
            # sample_weight=sample_weight,
            # weight_evaluation=weight_evaluation,
            # groups="group" if use_groups else None,
        ).fit(
            train_data=train_data[train_data["location"] == loc].
            ↪drop(columns=["ds"]),
            time_limit=time_limit,
            presets=presets,
            num_stack_levels=num_stack_levels,
            num_bag_folds=num_bag_folds if not use_groups else 2, # just put
            ↪somethin, will be overwritten anyways
            num_bag_sets=num_bag_sets,
            tuning_data=tuning_data[tuning_data["location"] == loc].
            ↪reset_index(drop=True).drop(columns=["ds"]) if use_tune_data else None,

```



```

        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
        excluded_model_types=excluded_model_types
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
        drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)

```

Warning: path already exists! This predictor may overwrite an existing predictor! path="AutogluonModels/submission_123_jorge_A"

Presets specified: ['best_quality']

Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8, num_bag_sets=20

Beginning AutoGluon training ... Time limit = 600s

AutoGluon will save models to "AutogluonModels/submission_123_jorge_A/"

AutoGluon Version: 0.8.1

Python Version: 3.10.12

Operating System: Darwin

Platform Machine: arm64

Platform Version: Darwin Kernel Version 22.1.0: Sun Oct 9 20:15:09 PDT 2022; root:xnu-8792.41.9~2/RELEASE_ARM64_T6000

Disk Space Avail: 131.19 GB / 494.38 GB (26.5%)

Train Data Rows: 30900

Train Data Columns: 44

Tuning Data Rows: 1504

Tuning Data Columns: 44

Label Column: y

Preprocessing data ...

AutoGluon infers your prediction problem is: 'regression' (because dtype of label-column == float and many unique label-values observed).

Label info (max, min, mean, stddev): (5733.42, 0.0, 674.06946, 1195.52285)

If 'regression' is not the correct problem_type, please manually specify the problem_type parameter during predictor init (You may specify problem_type as one of: ['binary', 'multiclass', 'regression'])

Using Feature Generators to preprocess the data ...

Fitting AutoMLPipelineFeatureGenerator...

```

Available Memory: 3016.0 MB
Train Data (Original) Memory Usage: 13.03 MB (0.4% of available memory)
Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
Stage 1 Generators:
    Fitting AsTypeFeatureGenerator...
        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
Stage 2 Generators:
    Fitting FillNaFeatureGenerator...
Stage 3 Generators:
    Fitting IdentityFeatureGenerator...
Stage 4 Generators:
    Fitting DropUniqueFeatureGenerator...
Stage 5 Generators:
    Fitting DropDuplicatesFeatureGenerator...
Useless Original Features (Count: 3): ['elevation:m', 'snow_drift:idx',
'location']
    These features carry no predictive signal and should be manually
investigated.
Training model for location A...

    This is typically a feature which has the same value for all
rows.

    These features do not need to be present at inference time.
Types of features in original data (raw dtype, special dtypes):
    ('float', []) : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]
    ('int', []) : 1 | ['is_estimated']
Types of features in processed data (raw dtype, special dtypes):
    ('float', []) : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]
    ('int', ['bool']) : 2 | ['snow_density:kgm3', 'is_estimated']
0.1s = Fit runtime
41 features in original data used to generate 41 features in processed
data.
Train Data (Processed) Memory Usage: 10.17 MB (0.3% of available memory)
Data preprocessing and feature engineering runtime = 0.13s ...
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
    This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
    To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:

```

```
{
    'NN_TORCH': {},
    'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}],
'GBMLarge'],
    'CAT': {},
    'XGB': {},
    'FASTAI': {},
    'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
    'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
    'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
```

Excluded models: ['CAT', 'XGB', 'RF'] (Specified by `excluded_model_types`)

Fitting 8 L1 models ...

Fitting model: KNeighborsUnif_BAG_L1 ... Training model for up to 599.87s of the 599.86s of remaining time.

-190.3604 = Validation score (-mean_absolute_error)

0.02s = Training runtime

119.71s = Validation runtime

Fitting model: KNeighborsDist_BAG_L1 ... Training model for up to 469.32s of the 469.32s of remaining time.

-191.9073 = Validation score (-mean_absolute_error)

0.02s = Training runtime

144.07s = Validation runtime

Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 313.98s of the 313.97s of remaining time.

Will use sequential fold fitting strategy because import of ray failed. Reason: ray is required to train folds in parallel. A quick tip is to install via `pip install ray==2.2.0`, or use sequential fold fitting by passing `sequential_local` to `ag_args_ensemble` when calling `tabular.fit` for example: `predictor.fit(..., ag_args_ensemble={'fold_fitting_strategy': 'sequential_local'})``

Fitting 8 child models (S1F1 - S1F8) | Fitting with SequentialLocalFoldFittingStrategy

[1000] valid_set's l1: 185.053

[2000] valid_set's l1: 178.583

[3000] valid_set's l1: 175.707

[4000] valid_set's l1: 173.75

[5000] valid_set's l1: 172.185

```

[6000] valid_set's l1: 171.275
[7000] valid_set's l1: 170.604

Ran out of time, early stopping on iteration 7828. Best iteration is:
[7827] valid_set's l1: 170.153

[1000] valid_set's l1: 193.64
[2000] valid_set's l1: 188.558
[3000] valid_set's l1: 185.938

Ran out of time, early stopping on iteration 3715. Best iteration is:
[3641] valid_set's l1: 184.827

[1000] valid_set's l1: 180.83
[2000] valid_set's l1: 175.11
[3000] valid_set's l1: 171.77
[4000] valid_set's l1: 170.097
[5000] valid_set's l1: 169.015
[6000] valid_set's l1: 168.407
[7000] valid_set's l1: 168.011
[8000] valid_set's l1: 167.395

Ran out of time, early stopping on iteration 8115. Best iteration is:
[8025] valid_set's l1: 167.363

[1000] valid_set's l1: 188.99
[2000] valid_set's l1: 183.143
[3000] valid_set's l1: 179.848
[4000] valid_set's l1: 178.118

Ran out of time, early stopping on iteration 4643. Best iteration is:
[4631] valid_set's l1: 177.531

[1000] valid_set's l1: 180.043
[2000] valid_set's l1: 174.672
[3000] valid_set's l1: 172.512
[4000] valid_set's l1: 170.112

Ran out of time, early stopping on iteration 4839. Best iteration is:
[4834] valid_set's l1: 168.704

[1000] valid_set's l1: 173.671
[2000] valid_set's l1: 169.207
[3000] valid_set's l1: 167.027

Ran out of time, early stopping on iteration 3760. Best iteration is:
[3666] valid_set's l1: 166.002

[1000] valid_set's l1: 189.372
[2000] valid_set's l1: 184.316
[3000] valid_set's l1: 182.143
[4000] valid_set's l1: 180.329
[5000] valid_set's l1: 179.437
[6000] valid_set's l1: 178.553

```

```

[7000] valid_set's l1: 178.221
[8000] valid_set's l1: 177.73

Ran out of time, early stopping on iteration 8677. Best iteration is:
[8656] valid_set's l1: 177.451

[1000] valid_set's l1: 182.091
[2000] valid_set's l1: 176.068
[3000] valid_set's l1: 172.904
[4000] valid_set's l1: 170.874
[5000] valid_set's l1: 169.534
[6000] valid_set's l1: 168.54
[7000] valid_set's l1: 167.609
[8000] valid_set's l1: 167.049

Ran out of time, early stopping on iteration 8332. Best iteration is:
[8332] valid_set's l1: 166.879
-81.5193          = Validation score    (-mean_absolute_error)
298.32s = Training    runtime
2.78s    = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 9.09s of the 9.09s
of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
    Ran out of time, early stopping on iteration 34. Best iteration is:
    [34] valid_set's l1: 296.732
    Ran out of time, early stopping on iteration 41. Best iteration is:
    [41] valid_set's l1: 273.849
    Ran out of time, early stopping on iteration 48. Best iteration is:
    [48] valid_set's l1: 245.869
    Ran out of time, early stopping on iteration 40. Best iteration is:
    [40] valid_set's l1: 281.07
    Ran out of time, early stopping on iteration 41. Best iteration is:
    [41] valid_set's l1: 263.936
    Ran out of time, early stopping on iteration 45. Best iteration is:
    [45] valid_set's l1: 248.645
    Ran out of time, early stopping on iteration 54. Best iteration is:
    [54] valid_set's l1: 241.851
    Ran out of time, early stopping on iteration 77. Best iteration is:
    [77] valid_set's l1: 210.857
    -176.5173          = Validation score    (-mean_absolute_error)
    8.72s    = Training    runtime
    0.03s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 ... Training model for up to 0.27s of the
0.27s of remaining time.
    -102.3261          = Validation score    (-mean_absolute_error)
    3.52s    = Training    runtime
    0.59s    = Validation runtime
Completed 1/20 k-fold bagging repeats ...

```

Fitting model: WeightedEnsemble_L2 ... Training model for up to 360.0s of the -4.4s of remaining time.

```
-81.5193      = Validation score    (-mean_absolute_error)
0.05s        = Training   runtime
0.0s         = Validation runtime
```

AutoGluon training complete, total runtime = 604.47s ... Best model:

"WeightedEnsemble_L2"

TabularPredictor saved. To load, use: predictor =

TabularPredictor.load("AutogluonModels/submission_123_jorge_A/")

Evaluation: mean_absolute_error on test data: -90.96713168170461

Note: Scores are always higher_is_better. This metric score can be multiplied by -1 to get the metric value.

Evaluations on test data:

```
{
  "mean_absolute_error": -90.96713168170461,
  "root_mean_squared_error": -243.00511998167275,
  "mean_squared_error": -59051.488337307164,
  "r2": 0.9243448828490051,
  "pearsonr": 0.961569748463051,
  "median_absolute_error": -5.074539422988892
}
```

Evaluation on test data:

-90.96713168170461

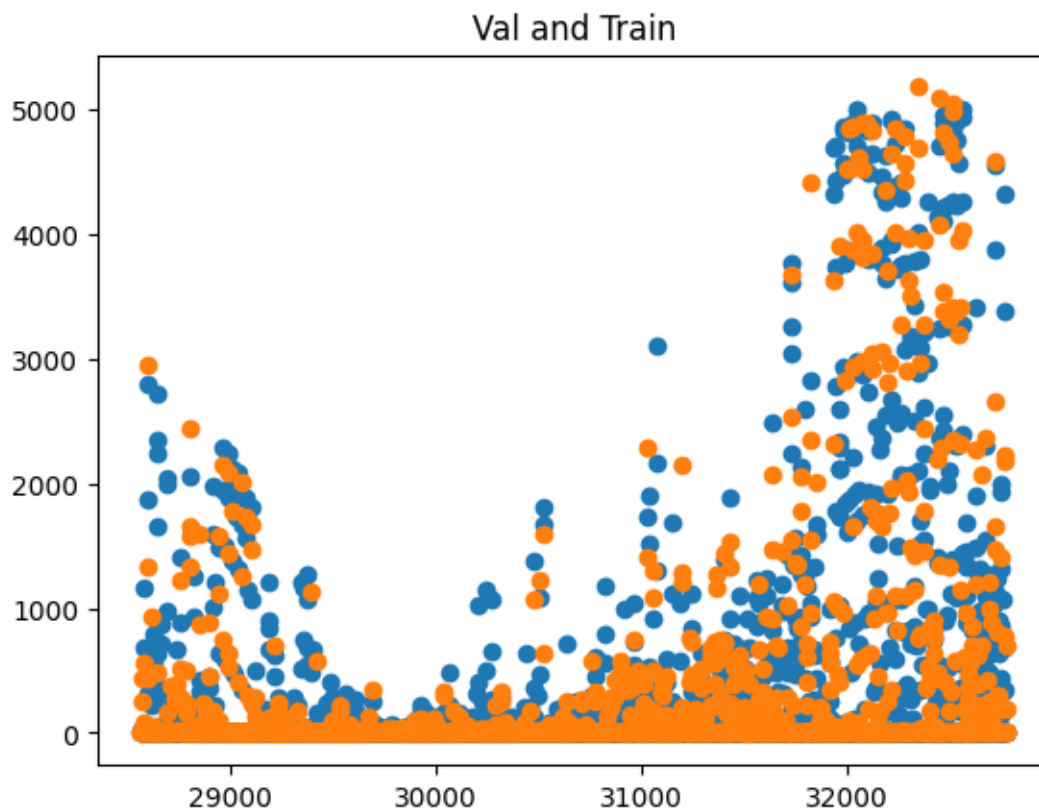
```
[67]: import matplotlib.pyplot as plt
leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_tune_data:
        plt.scatter(train_data[(train_data["location"] == loc) &
        ↪(train_data["is_estimated"]==True)]["y"].index,
        ↪train_data[(train_data["location"] == loc) &
        ↪(train_data["is_estimated"]==True)]["y"])
        plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,
        ↪tuning_data[tuning_data["location"] == loc]["y"])
        plt.title("Val and Train")
        plt.show()

    if use_test_data:
        lb = predictors[i].leaderboard(test_data[test_data["location"] ==
        ↪loc])
        lb["location"] = loc
        plt.scatter(test_data[test_data["location"] == loc]["y"].index,
        ↪test_data[test_data["location"] == loc]["y"])
        plt.title("Test")

    return lb
```

```
return pd.DataFrame()
```

```
leaderboards[0] = leaderboard_for_location(0, loc)
```

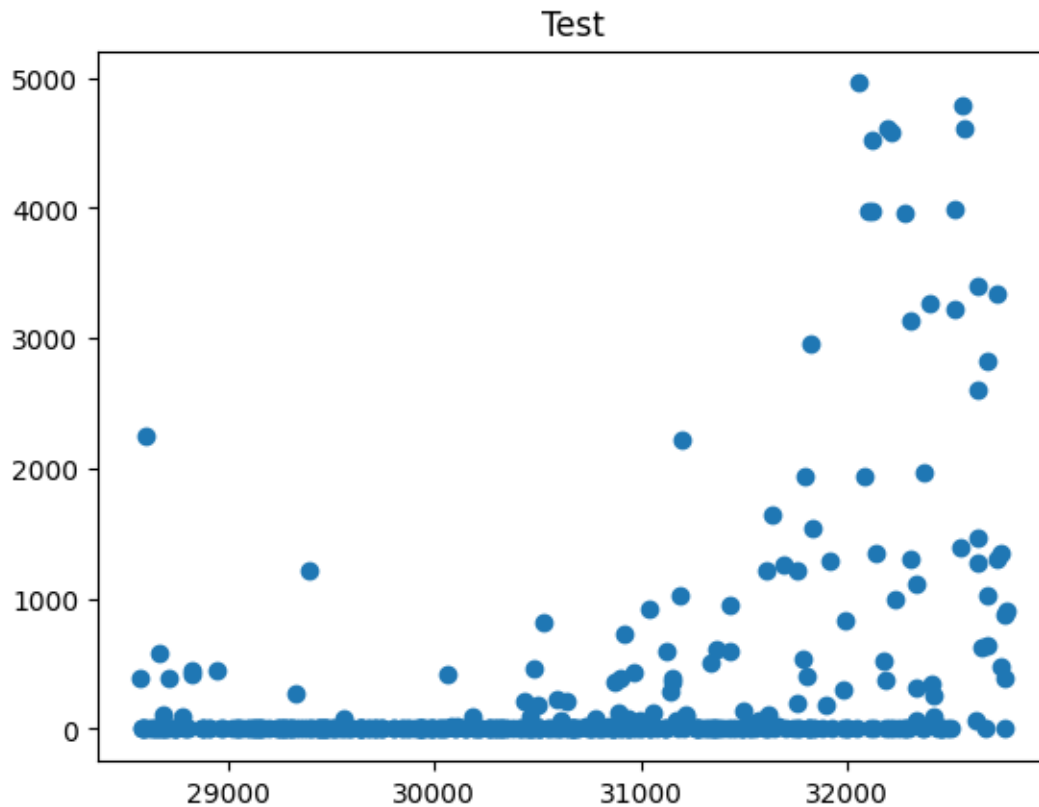


	model	score_test	score_val	pred_time_test	pred_time_val
fit_time	pred_time_test_marginal	pred_time_val_marginal	fit_time_marginal		
stack_level	can_infer	fit_order			
0	LightGBMXT_BAG_L1	-90.967132	-81.519269	0.610966	2.777259
298.320478		0.610966		2.777259	298.320478
1	True	3			
1	WeightedEnsemble_L2	-90.967132	-81.519269	0.612448	2.777490
298.370370		0.001482		0.000231	0.049892
2	True	6			
2	ExtraTreesMSE_BAG_L1	-103.476840	-102.326132	0.237853	0.588659
3.523309		0.237853		0.588659	3.523309
1	True	5			
3	LightGBM_BAG_L1	-165.818687	-176.517330	0.017727	0.030071
8.717963		0.017727		0.030071	8.717963
1	True	4			
4	KNeighborsUnif_BAG_L1	-170.349627	-190.360353	2.763736	119.714988
0.022840		2.763736		119.714988	0.022840

```

1      True      1
5  KNeighborsDist_BAG_L1 -176.177754 -191.907283      2.139242      144.067278
0.022986      2.139242      144.067278      0.022986
1      True      2

```



```

[68]: loc = "B"
predictors[1] = fit_predictor_for_location(loc)
leaderboards[1] = leaderboard_for_location(1, loc)

```

```

Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training ... Time limit = 600s
AutoGluon will save models to "AutogluonModels/submission_123_jorge_B/"
AutoGluon Version: 0.8.1
Python Version: 3.10.12
Operating System: Darwin
Platform Machine: arm64
Platform Version: Darwin Kernel Version 22.1.0: Sun Oct 9 20:15:09 PDT 2022;
root:xnu-8792.41.9~2/RELEASE_ARM64_T6000
Disk Space Avail: 130.64 GB / 494.38 GB (26.4%)
Train Data Rows: 27343

```



```

Train Data Columns: 44
Tuning Data Rows:    1504
Tuning Data Columns: 44
Label Column: y
Preprocessing data ...
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
    Label info (max, min, mean, stddev): (1152.3, -0.0, 97.86121, 206.22589)
    If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
    Available Memory:                2833.33 MB
    Train Data (Original) Memory Usage: 11.6 MB (0.4% of available memory)
    Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
    Stage 1 Generators:
        Fitting AsTypeFeatureGenerator...
            Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
    Stage 2 Generators:
        Fitting FillNaFeatureGenerator...
    Stage 3 Generators:
        Fitting IdentityFeatureGenerator...
    Stage 4 Generators:
        Fitting DropUniqueFeatureGenerator...
    Stage 5 Generators:
        Fitting DropDuplicatesFeatureGenerator...
    Useless Original Features (Count: 2): ['elevation:m', 'location']
    These features carry no predictive signal and should be manually
investigated.
        This is typically a feature which has the same value for all
rows.
        These features do not need to be present at inference time.
    Types of features in original data (raw dtype, special dtypes):
        ('float', []) : 41 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]
        ('int', [])   : 1 | ['is_estimated']
    Types of features in processed data (raw dtype, special dtypes):
        ('float', []) : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]
        ('int', ['bool']) : 2 | ['snow_density:kgm3', 'is_estimated']
    0.1s = Fit runtime
    42 features in original data used to generate 42 features in processed
data.

```

Train Data (Processed) Memory Usage: 9.29 MB (0.3% of available memory)
Data preprocessing and feature engineering runtime = 0.11s ...

Training model for location B...

AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'

This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.

To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).

User-specified model hyperparameters to be fit:

```
{
    'NN_TORCH': {},
    'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}],
    'GBMLarge'],
    'CAT': {},
    'XGB': {},
    'FASTAI': {},
    'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
    'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
    'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
```

Excluded models: ['CAT', 'XGB', 'RF'] (Specified by `excluded_model_types`)

Fitting 8 L1 models ...

Fitting model: KNeighborsUnif_BAG_L1 ... Training model for up to 599.89s of the
599.89s of remaining time.

```
-31.0941      = Validation score    (-mean_absolute_error)
0.02s        = Training    runtime
112.29s      = Validation runtime
```

Fitting model: KNeighborsDist_BAG_L1 ... Training model for up to 479.9s of the
479.9s of remaining time.

```
-30.6152      = Validation score    (-mean_absolute_error)
0.04s        = Training    runtime
117.99s      = Validation runtime
```

Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 353.48s of the
353.48s of remaining time.

Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy

[1000] valid_set's l1: 23.7938
[2000] valid_set's l1: 22.8059
[3000] valid_set's l1: 22.3022
[4000] valid_set's l1: 21.9856
[5000] valid_set's l1: 21.7541
[6000] valid_set's l1: 21.6093
[7000] valid_set's l1: 21.5328

Ran out of time, early stopping on iteration 7432. Best iteration is:
[7431] valid_set's l1: 21.4881

[1000] valid_set's l1: 25.6285
[2000] valid_set's l1: 24.3508
[3000] valid_set's l1: 23.7804
[4000] valid_set's l1: 23.4012
[5000] valid_set's l1: 23.2099
[6000] valid_set's l1: 23.0396

Ran out of time, early stopping on iteration 6588. Best iteration is:
[6587] valid_set's l1: 22.9706

[1000] valid_set's l1: 26.7749
[2000] valid_set's l1: 25.8799
[3000] valid_set's l1: 25.4253

Ran out of time, early stopping on iteration 3872. Best iteration is:
[3838] valid_set's l1: 25.2578

[1000] valid_set's l1: 25.4797
[2000] valid_set's l1: 24.4554
[3000] valid_set's l1: 24.0402
[4000] valid_set's l1: 23.85
[5000] valid_set's l1: 23.6779
[6000] valid_set's l1: 23.5788

Ran out of time, early stopping on iteration 6254. Best iteration is:
[6241] valid_set's l1: 23.5364

[1000] valid_set's l1: 24.2649
[2000] valid_set's l1: 23.4077

Ran out of time, early stopping on iteration 2922. Best iteration is:
[2920] valid_set's l1: 23.0069

[1000] valid_set's l1: 26.1133
[2000] valid_set's l1: 25.3134
[3000] valid_set's l1: 24.852
[4000] valid_set's l1: 24.5618
[5000] valid_set's l1: 24.4208

Ran out of time, early stopping on iteration 5783. Best iteration is:
[5757] valid_set's l1: 24.3455

```

[1000] valid_set's l1: 24.689
[2000] valid_set's l1: 23.8422
[3000] valid_set's l1: 23.548

Ran out of time, early stopping on iteration 3730. Best iteration is:
[3730] valid_set's l1: 23.3606

[1000] valid_set's l1: 25.663
[2000] valid_set's l1: 24.4793
[3000] valid_set's l1: 23.8906
[4000] valid_set's l1: 23.5761
[5000] valid_set's l1: 23.2969
[6000] valid_set's l1: 23.129
[7000] valid_set's l1: 22.9944
[8000] valid_set's l1: 22.8871
[9000] valid_set's l1: 22.8148

Ran out of time, early stopping on iteration 9319. Best iteration is:
[9319] valid_set's l1: 22.7999
-14.2659          = Validation score    (-mean_absolute_error)
336.66s = Training    runtime
2.29s    = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 11.37s of the 11.37s
of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
    Ran out of time, early stopping on iteration 84. Best iteration is:
    [84]    valid_set's l1: 27.6254
    Ran out of time, early stopping on iteration 72. Best iteration is:
    [72]    valid_set's l1: 32.1105
    Ran out of time, early stopping on iteration 63. Best iteration is:
    [63]    valid_set's l1: 33.7638
    Ran out of time, early stopping on iteration 72. Best iteration is:
    [72]    valid_set's l1: 31.0883
    Ran out of time, early stopping on iteration 61. Best iteration is:
    [61]    valid_set's l1: 31.6379
    Ran out of time, early stopping on iteration 59. Best iteration is:
    [59]    valid_set's l1: 33.0372
    Ran out of time, early stopping on iteration 54. Best iteration is:
    [54]    valid_set's l1: 34.1837
    Ran out of time, early stopping on iteration 90. Best iteration is:
    [90]    valid_set's l1: 30.0823
    -22.1354          = Validation score    (-mean_absolute_error)
    10.87s    = Training    runtime
    0.03s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 ... Training model for up to 0.37s of the
0.36s of remaining time.
    -16.7662          = Validation score    (-mean_absolute_error)
    3.1s        = Training    runtime

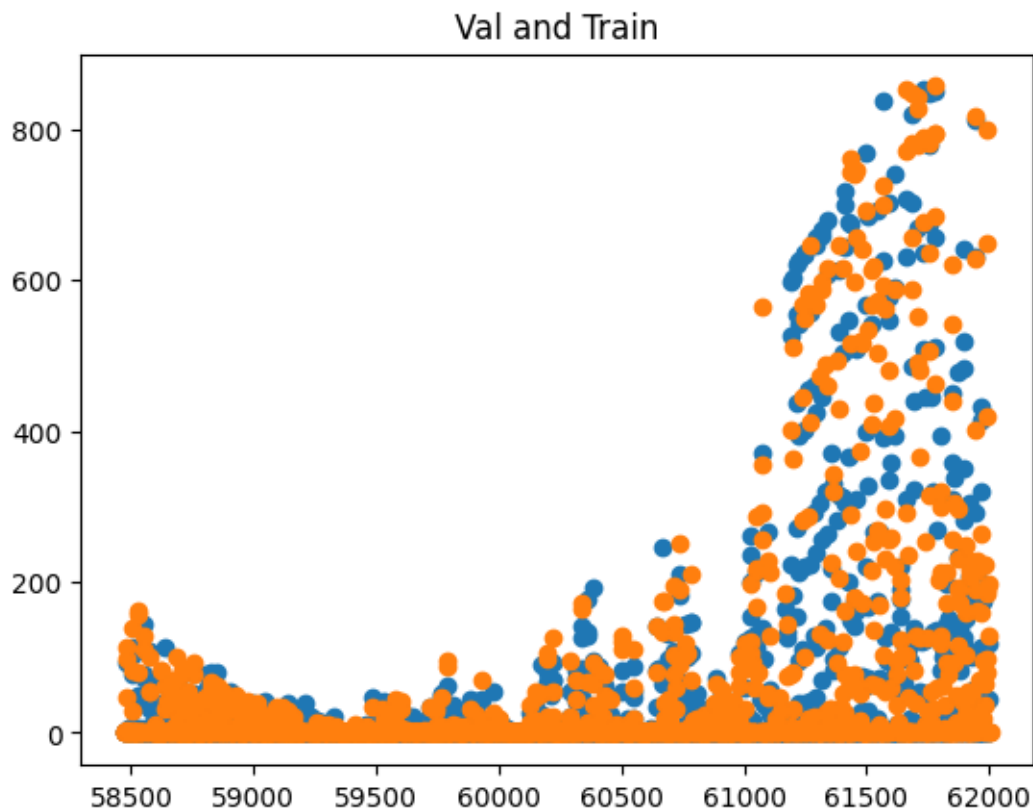
```

```

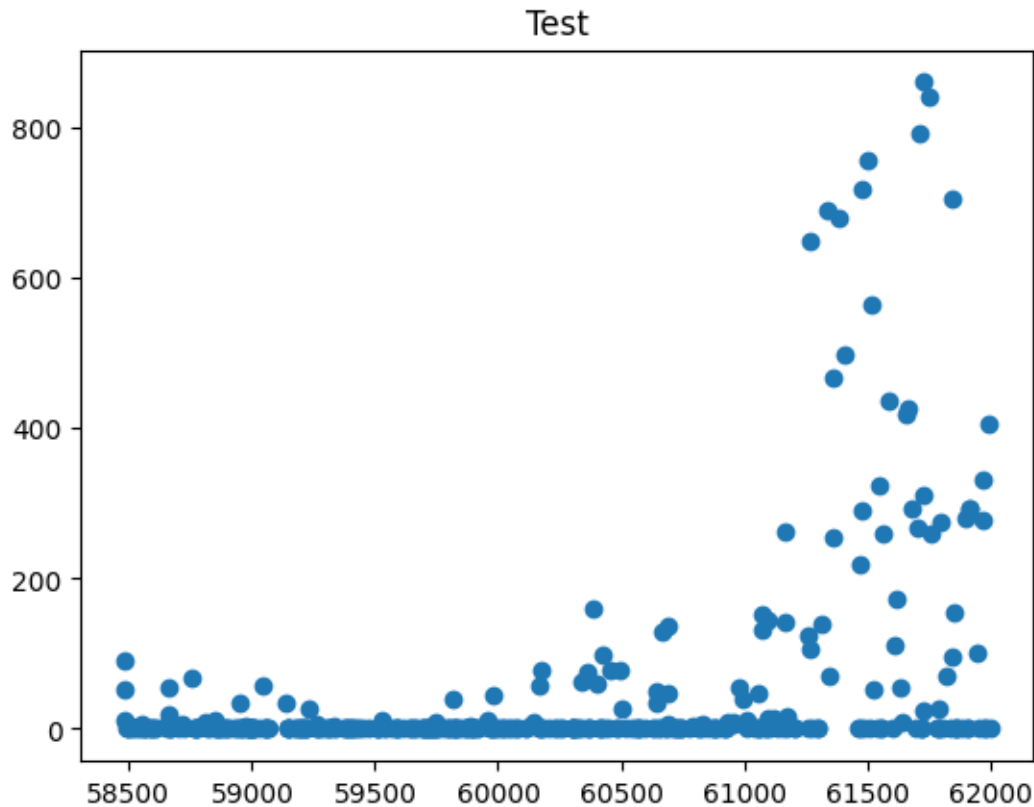
    0.52s    = Validation runtime
Completed 1/20 k-fold bagging repeats ...
Fitting model: WeightedEnsemble_L2 ... Training model for up to 360.0s of the
-3.73s of remaining time.
    -14.197 = Validation score    (-mean_absolute_error)
    0.05s   = Training    runtime
    0.0s    = Validation runtime
AutoGluon training complete, total runtime = 603.8s ... Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_123_jorge_B/")
Evaluation: mean_absolute_error on test data: -15.438844884969932
    Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -15.438844884969932,
    "root_mean_squared_error": -46.31067375306325,
    "mean_squared_error": -2144.678503462661,
    "r2": 0.8891341296924462,
    "pearsonr": 0.9429818273324595,
    "median_absolute_error": -0.5003813803195953
}

Evaluation on test data:
-15.438844884969932

```



	model	score_test	score_val	pred_time_test	pred_time_val
fit_time	pred_time_test_marginal	pred_time_val_marginal	fit_time_marginal		
stack_level	can_infer	fit_order			
0	WeightedEnsemble_L2	-15.438845	-14.197004	0.885067	2.806877
339.806925		0.001305		0.000248	0.052879
2	True	6			
1	LightGBMXT_BAG_L1	-15.720706	-14.265876	0.690143	2.286147
336.657248		0.690143		2.286147	336.657248
1	True	3			
2	ExtraTreesMSE_BAG_L1	-16.288915	-16.766233	0.193619	0.520482
3.096798		0.193619		0.520482	3.096798
1	True	5			
3	LightGBM_BAG_L1	-20.064119	-22.135438	0.018310	0.034770
10.872451		0.018310		0.034770	10.872451
1	True	4			
4	KNeighborsUnif_BAG_L1	-26.923013	-31.094123	1.672986	112.287918
0.017300		1.672986		112.287918	0.017300
1	True	1			
5	KNeighborsDist_BAG_L1	-27.258140	-30.615185	2.378423	117.987351
0.041716		2.378423		117.987351	0.041716
1	True	2			



```
[69]: loc = "C"
      predictors[2] = fit_predictor_for_location(loc)
      leaderboards[2] = leaderboard_for_location(2, loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training ... Time limit = 600s
AutoGluon will save models to "AutogluonModels/submission_123_jorge_C/"
AutoGluon Version: 0.8.1
Python Version: 3.10.12
Operating System: Darwin
Platform Machine: arm64
Platform Version: Darwin Kernel Version 22.1.0: Sun Oct 9 20:15:09 PDT 2022;
root:xnu-8792.41.9~2/RELEASE_ARM64_T6000
Disk Space Avail: 130.22 GB / 494.38 GB (26.3%)
Train Data Rows: 24034
Train Data Columns: 44
Tuning Data Rows: 1504
Tuning Data Columns: 44
Label Column: y
```

```

Preprocessing data ...
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
    Label info (max, min, mean, stddev): (999.6, -0.0, 81.13701, 169.91738)
    If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
    Available Memory:                2783.57 MB
    Train Data (Original) Memory Usage: 10.27 MB (0.4% of available memory)
    Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.

Training model for location C...

    Stage 1 Generators:
        Fitting AsTypeFeatureGenerator...
            Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
    Stage 2 Generators:
        Fitting FillNaFeatureGenerator...
    Stage 3 Generators:
        Fitting IdentityFeatureGenerator...
    Stage 4 Generators:
        Fitting DropUniqueFeatureGenerator...
    Stage 5 Generators:
        Fitting DropDuplicatesFeatureGenerator...
    Useless Original Features (Count: 3): ['elevation:m', 'snow_drift:idx',
'location']

        These features carry no predictive signal and should be manually
investigated.

        This is typically a feature which has the same value for all
rows.

        These features do not need to be present at inference time.
    Types of features in original data (raw dtype, special dtypes):
        ('float', []) : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]
        ('int', [])   : 1 | ['is_estimated']
    Types of features in processed data (raw dtype, special dtypes):
        ('float', []) : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]
        ('int', ['bool']) : 2 | ['snow_density:kgm3', 'is_estimated']
    0.1s = Fit runtime
    41 features in original data used to generate 41 features in processed
data.

    Train Data (Processed) Memory Usage: 8.02 MB (0.3% of available memory)

```



```

Data preprocessing and feature engineering runtime = 0.17s ...
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'

    This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.

    To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
    'NN_TORCH': {},
    'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}],
'GBMLarge'],
    'CAT': {},
    'XGB': {},
    'FASTAI': {},
    'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
    'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
    'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Excluded models: ['CAT', 'XGB', 'RF'] (Specified by `excluded_model_types`)
Fitting 8 L1 models ...
Fitting model: KNeighborsUnif_BAG_L1 ... Training model for up to 599.83s of the
599.83s of remaining time.
    -19.3805          = Validation score    (-mean_absolute_error)
    0.02s           = Training    runtime
    84.7s           = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 ... Training model for up to 506.64s of the
506.64s of remaining time.
    -19.5178          = Validation score    (-mean_absolute_error)
    0.02s           = Training    runtime
    96.04s          = Validation runtime
Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 402.86s of the
402.85s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy

[1000]  valid_set's l1: 20.4423
[2000]  valid_set's l1: 19.6976

```

```

[3000] valid_set's l1: 19.3299
[4000] valid_set's l1: 19.1266
[5000] valid_set's l1: 19.0207

Ran out of time, early stopping on iteration 5810. Best iteration is:
[5785] valid_set's l1: 18.9581

[1000] valid_set's l1: 19.5392
[2000] valid_set's l1: 18.7641
[3000] valid_set's l1: 18.4218
[4000] valid_set's l1: 18.2982

Ran out of time, early stopping on iteration 4682. Best iteration is:
[4681] valid_set's l1: 18.2238

[1000] valid_set's l1: 19.9367
[2000] valid_set's l1: 19.4343
[3000] valid_set's l1: 19.1701
[4000] valid_set's l1: 18.988

Ran out of time, early stopping on iteration 4442. Best iteration is:
[4441] valid_set's l1: 18.9296

[1000] valid_set's l1: 19.8456
[2000] valid_set's l1: 19.1611
[3000] valid_set's l1: 18.83

Ran out of time, early stopping on iteration 3493. Best iteration is:
[3493] valid_set's l1: 18.7373

[1000] valid_set's l1: 18.8303
[2000] valid_set's l1: 18.1495
[3000] valid_set's l1: 17.7981
[4000] valid_set's l1: 17.6383
[5000] valid_set's l1: 17.5538
[6000] valid_set's l1: 17.456

Ran out of time, early stopping on iteration 6401. Best iteration is:
[6384] valid_set's l1: 17.4308

[1000] valid_set's l1: 18.8522
[2000] valid_set's l1: 18.3947
[3000] valid_set's l1: 18.056
[4000] valid_set's l1: 17.9002

Ran out of time, early stopping on iteration 4233. Best iteration is:
[4206] valid_set's l1: 17.8586

[1000] valid_set's l1: 19.7356
[2000] valid_set's l1: 19.1368
[3000] valid_set's l1: 18.8649

Ran out of time, early stopping on iteration 3518. Best iteration is:
[3512] valid_set's l1: 18.7611

```

```
[1000] valid_set's l1: 19.6921
[2000] valid_set's l1: 19.0326
[3000] valid_set's l1: 18.6328
[4000] valid_set's l1: 18.4069
[5000] valid_set's l1: 18.2786
[6000] valid_set's l1: 18.1966
[7000] valid_set's l1: 18.1557
[8000] valid_set's l1: 18.1209
```

Ran out of time, early stopping on iteration 8550. Best iteration is:

```
[8521] valid_set's l1: 18.105
-11.7588      = Validation score    (-mean_absolute_error)
384.43s      = Training    runtime
1.81s        = Validation runtime
```

Fitting model: LightGBM_BAG_L1 ... Training model for up to 13.69s of the 13.68s of remaining time.

Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy

Ran out of time, early stopping on iteration 58. Best iteration is:

```
[58] valid_set's l1: 27.1011
```

Ran out of time, early stopping on iteration 80. Best iteration is:

```
[80] valid_set's l1: 22.8612
```

Ran out of time, early stopping on iteration 79. Best iteration is:

```
[79] valid_set's l1: 23.7087
```

Ran out of time, early stopping on iteration 87. Best iteration is:

```
[87] valid_set's l1: 22.5666
```

Ran out of time, early stopping on iteration 84. Best iteration is:

```
[84] valid_set's l1: 22.804
```

Ran out of time, early stopping on iteration 103. Best iteration is:

```
[103] valid_set's l1: 21.0775
```

Ran out of time, early stopping on iteration 102. Best iteration is:

```
[102] valid_set's l1: 22.6965
```

Ran out of time, early stopping on iteration 129. Best iteration is:

```
[129] valid_set's l1: 21.1514
```

```
-18.1958      = Validation score    (-mean_absolute_error)
```

```
13.12s      = Training    runtime
```

```
0.03s       = Validation runtime
```

Fitting model: ExtraTreesMSE_BAG_L1 ... Training model for up to 0.46s of the 0.46s of remaining time.

```
-15.6819      = Validation score    (-mean_absolute_error)
```

```
3.1s         = Training    runtime
```

```
0.44s        = Validation runtime
```

Completed 1/20 k-fold bagging repeats ...

Fitting model: WeightedEnsemble_L2 ... Training model for up to 360.0s of the -3.51s of remaining time.

```
-11.7568      = Validation score    (-mean_absolute_error)
```

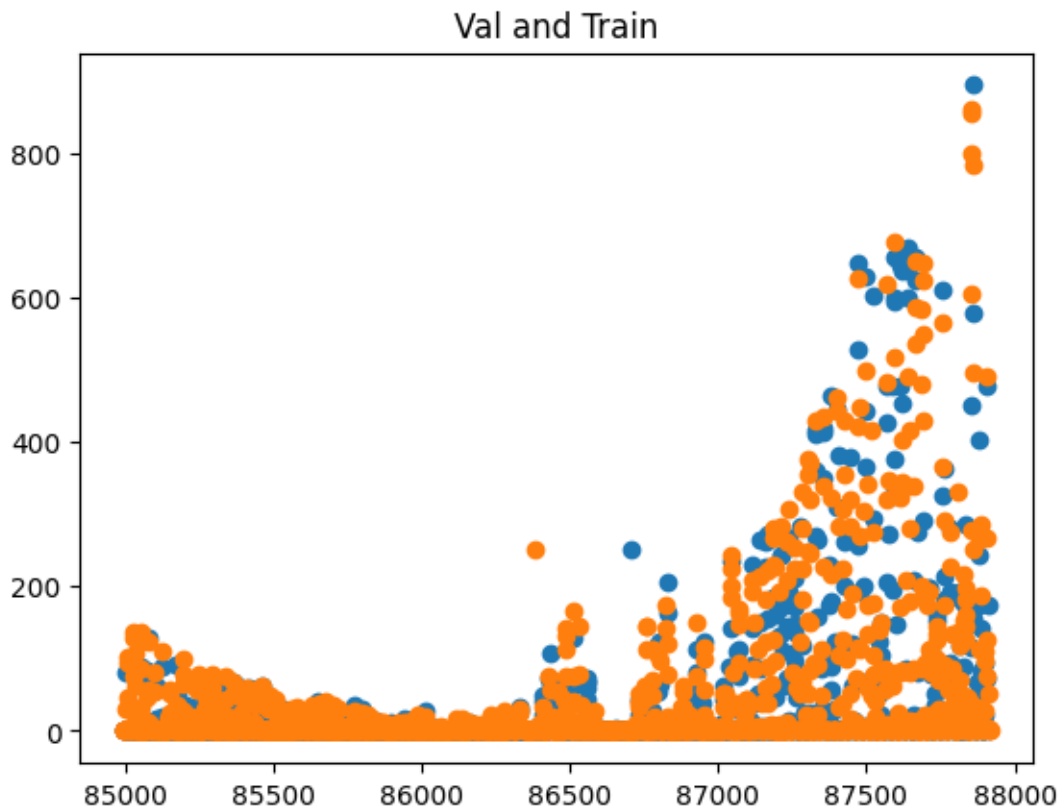
```
0.05s        = Training    runtime
```

```
0.0s         = Validation runtime
```

```

AutoGluon training complete, total runtime = 603.59s ... Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_123_jorge_C/")
Evaluation: mean_absolute_error on test data: -13.096481825131798
    Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
  "mean_absolute_error": -13.096481825131798,
  "root_mean_squared_error": -34.23995914165886,
  "mean_squared_error": -1172.374802022468,
  "r2": 0.8864144040338552,
  "pearsonr": 0.952584961559198,
  "median_absolute_error": -0.82662034034729
}
Evaluation on test data:
-13.096481825131798

```

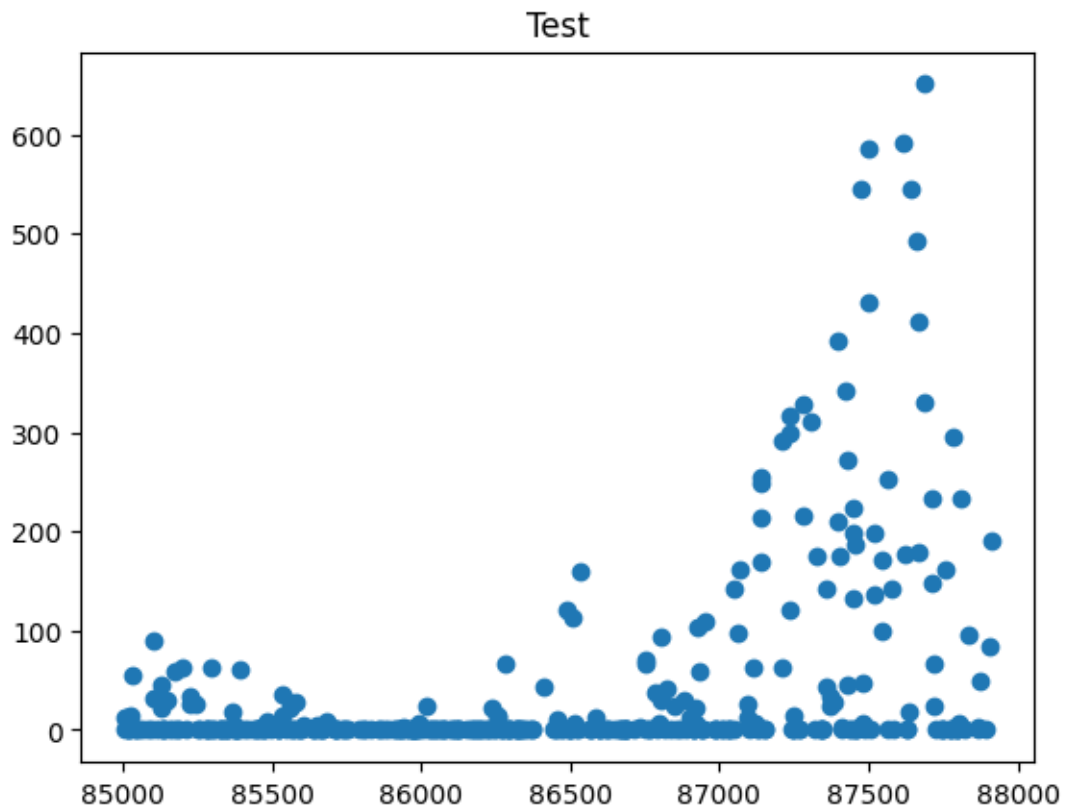


```

          model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal

```

stack_level	can_infer	fit_order				
0	WeightedEnsemble_L2	-13.096482	-11.756789	1.870625	86.513985	
		384.506397	0.001869	0.000482	0.053767	
2	True	6				
1	LightGBMXT_BAG_L1	-13.150320	-11.758754	0.541423	1.814085	
		384.431918	0.541423	1.814085	384.431918	
1	True	3				
2	ExtraTreesMSE_BAG_L1	-17.003587	-15.681893	0.157400	0.436322	
		3.095096	0.157400	0.436322	3.095096	
1	True	5				
3	KNeighborsDist_BAG_L1	-18.491110	-19.517754	1.457215	96.043076	
		0.018904	1.457215	96.043076	0.018904	
1	True	2				
4	KNeighborsUnif_BAG_L1	-18.537639	-19.380456	1.327333	84.699418	
		0.020712	1.327333	84.699418	0.020712	
1	True	1				
5	LightGBM_BAG_L1	-20.982739	-18.195840	0.017541	0.029602	
		13.117195	0.017541	0.029602	13.117195	
1	True	4				



```
[70]: # save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")

for i in range(len(predictors)):
    print(f"Predictor {i}:")
    print(predictors[i].
    ↪info()["model_info"]["WeightedEnsemble_L2"]["children_info"]["S1F1"]["model_weights"])
```

```
Predictor 0:
{'LightGBMXT_BAG_L1': 1.0}
Predictor 1:
{'LightGBMXT_BAG_L1': 0.8481012658227848, 'ExtraTreesMSE_BAG_L1':
0.1518987341772152}
Predictor 2:
{'KNeighborsUnif_BAG_L1': 0.015873015873015872, 'LightGBMXT_BAG_L1':
0.9841269841269841}
```

5 Submit

```
[71]: import pandas as pd
import matplotlib.pyplot as plt

future_test_data = TabularDataset('X_test_raw.csv')
future_test_data["ds"] = pd.to_datetime(future_test_data["ds"])
#test_data
```

Loaded data from: X_test_raw.csv | Columns = 45 / 45 | Rows = 4608 -> 4608

```
[72]: test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
future_test_data_merged = pd.merge(future_test_data, test_ids, how="inner",
    ↪right_on=["time", "location"], left_on=["ds", "location"])

#test_data_merged
```

Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160

```
[73]: # predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in future_test_data.groupby('location'):
    i = location_map[loc]
```

```

subset = future_test_data_merged[future_test_data_merged["location"] == loc]
↳loc].reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    #train_data.loc[train_data["location"] == loc, "prediction"] = 
↳predictors[i].predict(train_data[train_data["location"] == loc])
    if use_tune_data:
        tuning_data.loc[tuning_data["location"] == loc, "prediction"] = 
↳predictors[i].predict(tuning_data[tuning_data["location"] == loc])
    if use_test_data:
        test_data.loc[test_data["location"] == loc, "prediction"] = 
↳predictors[i].predict(test_data[test_data["location"] == loc])

```

```

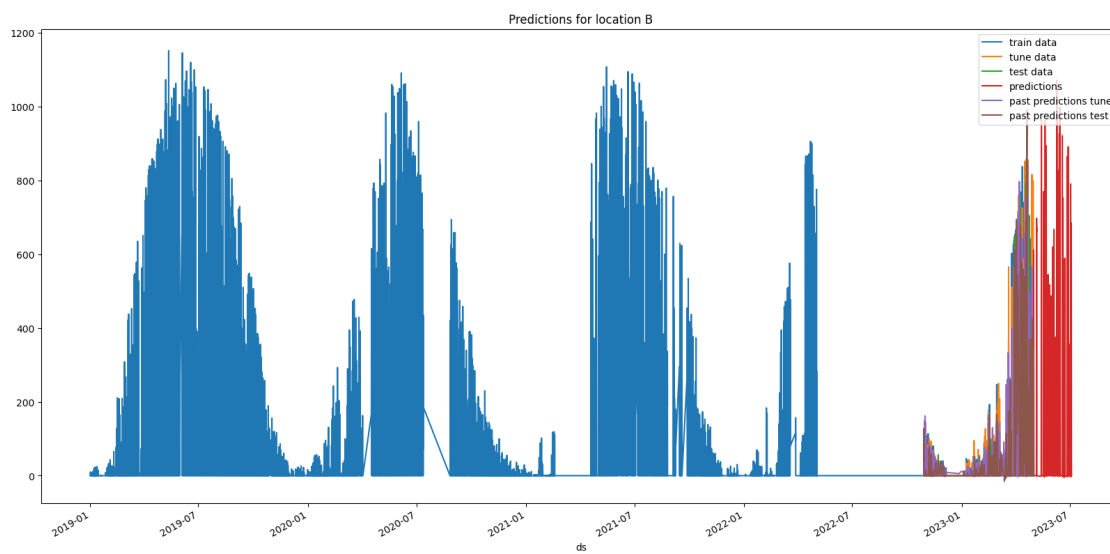
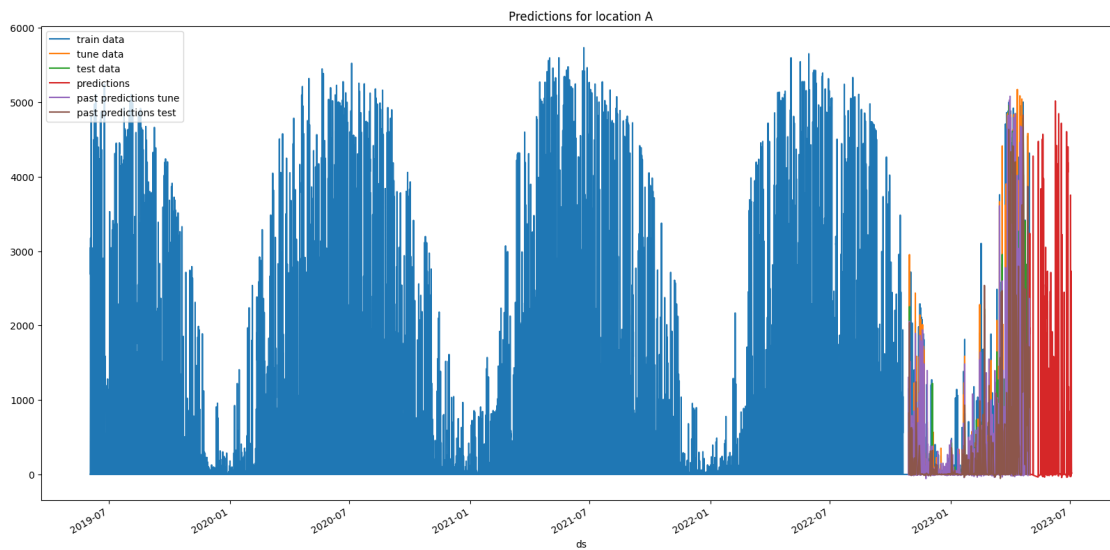
[74]: # plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data[train_data["location"]==loc].plot(x='ds', y='y', ax=ax, 
↳label="train data")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='y', ax=ax, 
↳label="tune data")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='y', ax=ax, 
↳label="test data")

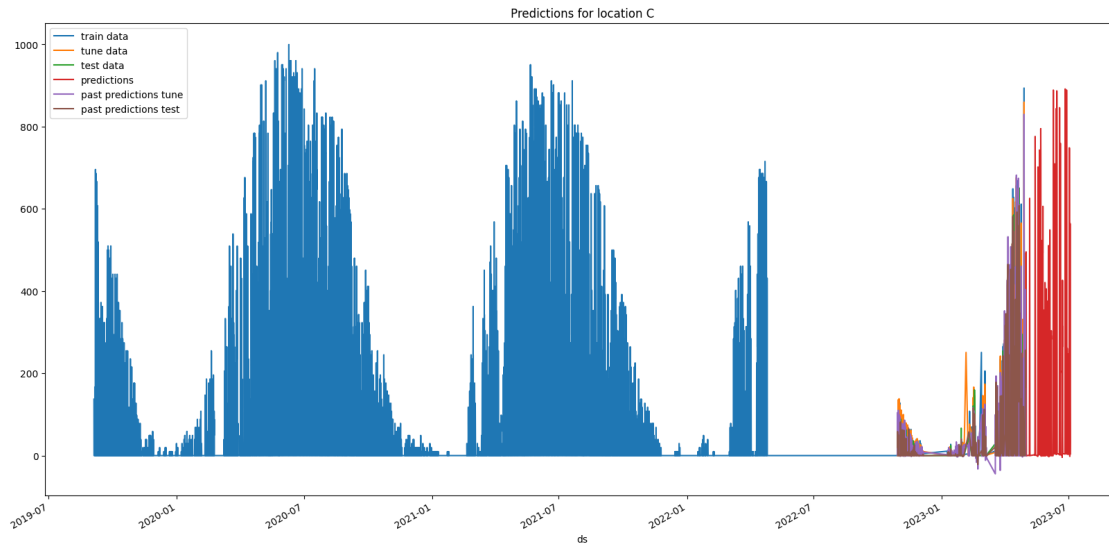
    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    #train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds', 
↳y='prediction', ax=ax, label="past predictions")
    #train_data[train_data["location"]==loc].plot(x='ds', y='prediction', 
↳ax=ax, label="past predictions train")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='prediction', 
↳ax=ax, label="past predictions tune")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='prediction', 
↳ax=ax, label="past predictions test")

```

```
# title
ax.set_title(f"Predictions for location {loc}")
```





```
[75]: temp_predictions = [prediction.copy() for prediction in predictions]
if clip_predictions:
    # clip predictions smaller than 0 to 0
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())

# Instead of clipping, shift all prediction values up by the largest negative
↪number.
# This way, the smallest prediction will be 0.
elif shift_predictions:
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred["prediction"] = pred["prediction"] - pred["prediction"].min()
        print("Smallest prediction after clipping:", pred["prediction"].min())

elif shift_predictions_by_average_of_negatives_then_clip:
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        mean_negative = pred[pred["prediction"] < 0]["prediction"].mean()
        # if not nan
        if mean_negative == mean_negative:
            pred["prediction"] = pred["prediction"] - mean_negative
```

```

pred.loc[pred["prediction"] < 0, "prediction"] = 0
print("Smallest prediction after clipping:", pred["prediction"].min())

```

```

# concatenate predictions
submissions_df = pd.concat(temp_predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df

```

```

Smallest prediction: -41.899334
Smallest prediction after clipping: 0.0
Smallest prediction: -4.4888353
Smallest prediction after clipping: 0.0
Smallest prediction: -4.3920894
Smallest prediction after clipping: 0.0

```

```

[75]:      id  prediction
0      0      0.000000
1      1      0.000000
2      2      0.000000
3      3     45.446655
4      4    262.431946
..    ...      ...
715   2155    70.653954
716   2156    45.066662
717   2157    11.451385
718   2158     4.537232
719   2159     4.065796

```

```

[2160 rows x 2 columns]

```

```

[76]: # Save the submission DataFrame to submissions folder, create new name based on
      ↪ last submission, format is submission_<last_submission_number + 1>.csv

      # Save the submission
      print(f"Saving submission to submissions/{new_filename}.csv")
      submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),
      ↪ index=False)
      print("jall1a")

```

```

Saving submission to submissions/submission_123_jorge.csv
jall1a

```

```

[77]: # feature importance
      # print starting calculating feature importance for location A with big text
      ↪ font

```

```

print("\033[1m" + "Calculating feature importance for location A..." +
      "\033[0m")
predictors[0].feature_importance(feature_stage="original",
      data=test_data[test_data["location"] == "A"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location B..." +
      "\033[0m")
predictors[1].feature_importance(feature_stage="original",
      data=test_data[test_data["location"] == "B"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location C..." +
      "\033[0m")
predictors[2].feature_importance(feature_stage="original",
      data=test_data[test_data["location"] == "C"], time_limit=60*10)

```

These features in provided data are not utilized by the predictor and will be ignored: ['ds', 'elevation:m', 'snow_drift:idx', 'location', 'prediction']
 Computing feature importance via permutation shuffling for 41 features using 376 rows with 10 shuffle sets... Time limit: 600s...

Calculating feature importance for location A...

270.44s = Expected runtime (27.04s per shuffle set)

103.13s = Actual runtime (Completed 10 of 10 shuffle sets)

These features in provided data are not utilized by the predictor and will be ignored: ['ds', 'elevation:m', 'location', 'prediction']

Computing feature importance via permutation shuffling for 42 features using 376 rows with 10 shuffle sets... Time limit: 600s...

Calculating feature importance for location B...

409.32s = Expected runtime (40.93s per shuffle set)

```

-----
KeyboardInterrupt                                Traceback (most recent call last)
/Users/jorgensandhaug/Desktop/tdt4173/TDT4173/autogluon_each_location.ipynb Cell
↳ 37 line 6
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/
      ↳ TDT4173/autogluon_each_location.ipynb#Y225sZmlsZQ%3D%3D?line=3'>4</a>
      ↳ predictors[0].feature_importance(feature_stage="original",
      ↳ data=test_data[test_data["location"] == "A"], time_limit=60*10)
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/
      ↳ TDT4173/autogluon_each_location.ipynb#Y225sZmlsZQ%3D%3D?line=4'>5</a>
      ↳ print("\033[1m" + "Calculating feature importance for location B..." + "\033[0m" )
----> <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/
      ↳ TDT4173/autogluon_each_location.ipynb#Y225sZmlsZQ%3D%3D?line=5'>6</a>
      ↳ predictors[1].feature_importance(feature_stage="original",
      ↳ data=test_data[test_data["location"] == "B"], time_limit=60*10)
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/
      ↳ TDT4173/autogluon_each_location.ipynb#Y225sZmlsZQ%3D%3D?line=6'>7</a>
      ↳ print("\033[1m" + "Calculating feature importance for location C..." + "\033[0m" )

```

```

    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/
    ↳TDT4173/autogluon_each_location.ipynb#Y225sZmlsZQ%3D%3D?line=7'>8</a>
    ↳predictors[2].feature_importance(feature_stage="original",
    ↳data=test_data[test_data["location"] == "C"], time_limit=60*10)

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/
↳tabular/predictor/predictor.py:2425, in TabularPredictor.
↳feature_importance(self, data, model, features, feature_stage, subsample_size,
↳time_limit, num_shuffle_sets, include_confidence_band, confidence_level,
↳silent)

```

```

    2422 if num_shuffle_sets is None:
    2423     num_shuffle_sets = 10 if time_limit else 5
-> 2425 fi_df = self._learner.get_feature_importance(
    2426     model=model,
    2427     X=data,
    2428     features=features,
    2429     feature_stage=feature_stage,
    2430     subsample_size=subsample_size,
    2431     time_limit=time_limit,
    2432     num_shuffle_sets=num_shuffle_sets,
    2433     silent=silent,
    2434 )
    2436 if include_confidence_band:
    2437     if confidence_level <= 0.5 or confidence_level >= 1.0:

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/
↳tabular/learner/abstract_learner.py:870, in AbstractTabularLearner.
↳get_feature_importance(self, model, X, y, features, feature_stage,
↳subsample_size, silent, **kwargs)

```

```

    867     X = X.drop(columns=unused_features)
    869     if feature_stage == "original":
--> 870     return trainer._get_feature_importance_raw(
    871         model=model, X=X, y=y, features=features,
    ↳subsample_size=subsample_size, transform_func=self.transform_features,
    ↳silent=silent, **kwargs
    872     )
    873     X = self.transform_features(X)
    874 else:

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/core /
↳trainer/abstract_trainer.py:2574, in AbstractTrainer.
↳get_feature_importance_raw(self, X, y, model, eval_metric, **kwargs)

```

```

    2572 model: AbstractModel = self.load_model(model)
    2573 predict_func_kwargs = dict(model=model)
-> 2574 return compute_permutation_feature_importance(
    2575     X=X,
    2576     y=y,
    2577     predict_func=predict_func,
    2578     predict_func_kwargs=predict_func_kwargs,
    2579     eval_metric=eval_metric,

```

```

2580     quantile_levels=self.quantile_levels,
2581     **kwargs,
2582 )

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/core /
↳ utils/utils.py:867, in compute_permutation_feature_importance(X, y,
↳ predict_func, eval_metric, features, subsample_size, num_shuffle_sets,
↳ predict_func_kwargs, transform_func, transform_func_kwargs, time_limit,
↳ silent, log_prefix, importance_as_list, random_state, **kwargs)

```

```

    865 else:
    866     X_raw_transformed = X_raw if transform_func is None else
↳ transform_func(X_raw, **transform_func_kwargs)
--> 867 y_pred = predict_func(X_raw_transformed, **predict_func_kwargs)
    869 row_index = 0
    870 for feature in parallel_computed_features:

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/core /
↳ trainer/abstract_trainer.py:749, in AbstractTrainer.predict(self, X, model)

```

```

    747     model = self._get_best()
    748     cascade = isinstance(model, list)
--> 749 return self._predict_model(X, model, cascade=cascade)

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/core /
↳ trainer/abstract_trainer.py:2388, in AbstractTrainer._predict_model(self, X,
↳ model, model_pred_proba_dict, cascade)

```

```

    2387 def _predict_model(self, X, model, model_pred_proba_dict=None,
↳ cascade=False):
-> 2388     y_pred_proba = self._predict_proba_model(X=X, model=model,
↳ model_pred_proba_dict=model_pred_proba_dict, cascade=cascade)
    2389     return get_pred_from_proba(y_pred_proba=y_pred_proba,
↳ problem_type=self.problem_type)

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/core /
↳ trainer/abstract_trainer.py:2392, in AbstractTrainer.
↳ _predict_proba_model(self, X, model, model_pred_proba_dict, cascade)

```

```

    2391 def _predict_proba_model(self, X, model, model_pred_proba_dict=None,
↳ cascade=False):
-> 2392     return self.get_pred_proba_from_model(model=model, X=X,
↳ model_pred_proba_dict=model_pred_proba_dict, cascade=cascade)

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/core /
↳ trainer/abstract_trainer.py:769, in AbstractTrainer.
↳ get_pred_proba_from_model(self, model, X, model_pred_proba_dict, cascade)

```

```

    767 else:
    768     models = [model]
--> 769 model_pred_proba_dict = self.get_model_pred_proba_dict(X=X,
↳ models=models, model_pred_proba_dict=model_pred_proba_dict, cascade=cascade)
    770 if not isinstance(model, str):
    771     model = model.name

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/core /
↳ trainer/abstract_trainer.py:1018, in AbstractTrainer.
↳ get_model_pred_proba_dict(self, X, models, model_pred_proba_dict,
↳ model_pred_time_dict, record_pred_time, use_val_cache, cascade,
↳ cascade_threshold)

```

```

    1016     else:
    1017         preprocess_kwargs = dict(infer=False,
↳ model_pred_proba_dict=model_pred_proba_dict)
-> 1018     model_pred_proba_dict[model_name] = model.predict_proba(X,
↳ **preprocess_kwargs)
    1019 else:
    1020     model_pred_proba_dict[model_name] = model.predict_proba(X)

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/core /
↳ models/ensemble/bagged_ensemble_model.py:346, in BaggedEnsembleModel.
↳ predict_proba(self, X, normalize, **kwargs)

```

```

    344 model = self.load_child(self.models[0])
    345 X = self.preprocess(X, model=model, **kwargs)
--> 346 pred_proba = model.predict_proba(X=X, preprocess_nonadaptive=False,
↳ normalize=normalize)
    347 for model in self.models[1:]:
    348     model = self.load_child(model)

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/core /
↳ models/abstract/abstract_model.py:931, in AbstractModel.predict_proba(self, X,
↳ normalize, **kwargs)

```

```

    929 if normalize is None:
    930     normalize = self.normalize_pred_proba
--> 931 y_pred_proba = self._predict_proba(X=X, **kwargs)
    932 if normalize:
    933     y_pred_proba = normalize_pred_proba(y_pred_proba, self.problem_type)

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon /
↳ tabular/models/lgb/lgb_model.py:234, in LGBModel._predict_proba(self, X,
↳ num_cpus, **kwargs)

```

```

    231 def _predict_proba(self, X, num_cpus=0, **kwargs):
    232     X = self.preprocess(X, **kwargs)
--> 234     y_pred_proba = self.model.predict(X, num_threads=num_cpus)
    235     if self.problem_type == REGRESSION:
    236         return y_pred_proba

```

```

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/lightgbm/basi .
↳ py:3538, in Booster.predict(self, data, start_iteration, num_iteration,
↳ raw_score, pred_leaf, pred_contrib, data_has_header, is_reshape, **kwargs)

```

```

    3536     else:
    3537         num_iteration = -1
-> 3538 return predictor.predict(data, start_iteration, num_iteration,
    3539                          raw_score, pred_leaf, pred_contrib,

```

```
3540 data_has_header, is_reshape)
```

```
File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/lightgbm/basic.py:848, in _InnerPredictor.predict(self, data, start_iteration, num_iteration, raw_score, pred_leaf, pred_contrib, data_has_header, is_reshape)
```

```
846 preds, nrow = self.__pred_for_csc(data, start_iteration, num_iteration, predict_type)
847 elif isinstance(data, np.ndarray):
--> 848 preds, nrow = self.__pred_for_np2d(data, start_iteration, num_iteration, predict_type)
849 elif isinstance(data, list):
850     try:
```

```
File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/lightgbm/basic.py:938, in _InnerPredictor.__pred_for_np2d(self, mat, start_iteration, num_iteration, predict_type)
```

```
936 return preds, nrow
937 else:
--> 938 return inner_predict(mat, start_iteration, num_iteration, predict_type)
```

```
File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/lightgbm/basic.py:908, in _InnerPredictor.__pred_for_np2d.<locals>.inner_predict(mat, start_iteration, num_iteration, predict_type, preds)
```

```
906 raise ValueError("Wrong length of pre-allocated predict array")
907 out_num_preds = ctypes.c_int64(0)
--> 908 _safe_call(_LIB.LGBM_BoosterPredictForMat(
909     self.handle,
910     ptr_data,
911     ctypes.c_int(type_ptr_data),
912     ctypes.c_int32(mat.shape[0]),
913     ctypes.c_int32(mat.shape[1]),
914     ctypes.c_int(C_API_IS_ROW_MAJOR),
915     ctypes.c_int(predict_type),
916     ctypes.c_int(start_iteration),
917     ctypes.c_int(num_iteration),
918     c_str(self.pred_parameter),
919     ctypes.byref(out_num_preds),
920     preds.ctypes.data_as(ctypes.POINTER(ctypes.c_double))))
921 if n_preds != out_num_preds.value:
922     raise ValueError("Wrong length for predict results")
```

KeyboardInterrupt:

```
[ ]: # save this notebook to submissions folder
import subprocess
import os
```

```
# subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
    ↳ join('notebook_pdfs', f"{new_filename}_automatic_save.pdf"),
    ↳ "autogluon_each_location.ipynb"])
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
    ↳ join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
    ↳ ipynb"])
```

```
[ ]: # import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,
    ↳ stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
    ↳ strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',
    ↳ 'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is
    ↳ not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b', branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m', commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',
    ↳ 'origin', branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
```



```
#     execute_git_command(git_repo_path, ['push', '--set-upstream',  
↪ 'origin', branch_name])  
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])  
  
# execute_git_command(git_repo_path, ['checkout', 'main'])
```

[]: